Modeling Relationships Using Graph State Variables[1]

Matthew B. Bennett and Robert D. Rasmussen
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109
(818) 393-0836
Matthew.B.Bennett@jpl.nasa.gov and Robert.D.Rasmussen@jpl.nasa.gov

Abstract—The Mission Data System is a unified flight, ground, simulation, and test software system for space missions. Its first application will be the Mars '09 mission, where common MDS software frameworks will be adapted for use in interplanetary cruise, entry-descent-landing, and rover operations. A key architectural theme of MDS is explicit modeling of states. This provides a sound foundation for estimation, control, and data analysis. Certain essential states (e.g. attitude and location) are relative rather than absolute. Relative states are defined in graph state variables (GSVs) as relationships between nodes in a graph. GSVS are a general graph based state representation that (1) can derive a state's value by combining relationships, (2) produces different results for different derivation paths, (3) handles changes to topology and relationships between nodes, and (4) represents dependencies between relationships (e.g. correlations). This paper shows example GSV representations for spacecraft orientation, location, trajectories, dynamics, and kinematics.

TABLE OF CONTENTS

1. INTRODUCTION

Future space missions are becoming more challenging and complex. Flybys of planets are now being followed up with landers having in-situ mobile vehicles. Mobile vehicles have demanding objectives to achieve scientific objectives without the intervention of ground controllers. Obstacles must be avoided. Chance objects of interest need to be efficiently identified and investigated on the way to achieving specific ground directed mission objectives. In some cases, timely communications with Earth may be impossible because of the long distances involved, or because of blocked communications. Flybys themselves are becoming more challenging. The spacecraft must compensate for uncertainties in small body trajectories that cannot be anticipated by their human operators on Earth, yet require accurate identification and targeting to meet mission objectives. Missions that previously could be planned in advance as detailed sequence of commands now must be flexible to handle uncertainties in time, trajectory, and science opportunities. The availability of more capable and affordable computing power enables these mission objectives. But this also puts more responsibility on software to implement more advanced capabilities. Further, advances in microspacecraft technology, now make it affordable to launch many more spacecraft than in the past and conceive of missions that are comprised of fleet of cooperating spacecraft, putting further demands on spacecraft operations. Mission operations for this next wave of spacecraft will be prohibitively expensive if we continue to orchestrate every detail of spacecraft commanding using human intensive sequencing tools and processes. Clearly, there is a need for more spacecraft autonomy in the next generation of spacecraft to simplify human-spacecraft interactions.

Software costs will limit the potential for taking advantage of these new opportunities. Currently, spacecraft software is custom designed and built based on inherited capabilities, standards, and operational paradigms from previous missions. However, much usually has to be re-written and validated. Many assumptions about the spacecraft and mission are hidden within the software code without an abstraction that facilitates reuse. Time and again it turns out to be cheaper to rewrite rather than reuse. Even when the software is well documented, the assumptions are too tightly coupled with the original design for reuse without significant rework.

The Mission Data System (MDS) is a unified flight, ground, and test architecture under development at the Jet Propulsion Laboratory that provides a foundation for the autonomous missions of NASA's future. The architecture provides a system engineering methodology for identifying

[1] 0-7803-6599-2/01/$10.00 © 2001 IEEE

the mission assumptions. The methodology also provides for direct translation of the assumptions into specific software architectural elements. These explicit models of assumptions within the software are easily inspected and modified from mission to mission. The software architectural elements are generic frameworks built for reuse and adaptation for each new mission. Future mission software specialists will be able to concentrate on new autonomous capabilities of the future rather than re-engineering software to achieve the same objectives of previous missions.

This paper describes one of the general MDS software frameworks for modeling assumptions. MDS has a number of architectural patterns for modeling assumptions. One pattern is the explicit representation of state as a function of time. Many states are *absolute* (e.g. volume, pressure, current) and can be modeled as scalars with units. Others are *relative* and are defined with respect to a reference (e.g. voltage with respect to a ground, location with respect to an origin, orientation with respect to a coordinate frame). Graph state variables are both a software architecture framework and a means for expressing systems engineering concepts for composable time-varying relationships.

2. MDS Overview

The Mission Data System vision is that software plays a central and increasingly important system role that must be reconciled with the overall systems engineering approach adopted by a project. Both software and systems engineering apply across all parts of a project and to all elements of the environment affecting the mission. Therefore, it is essential that the systems and software share a common approach to defining, describing, developing, understanding, testing, operating, and visualizing what systems do. To meet this vision, MDS is founded on a set of architectural themes that shape the design [1]. These themes are emphasized because they have a broad impact on the design and they differ from earlier spacecraft engineering practices. Key themes that are manifested by graph state variables are described below.

State and Models Are Central

MDS is founded upon a state-based architecture, where *state* is a representation of a momentary condition of an evolving system and *models* describe how states evolve. State is accessible in a uniform way through *state variables*, as apposed to a program's local variables. State evolution is described on *state timelines*, which are a complete record of

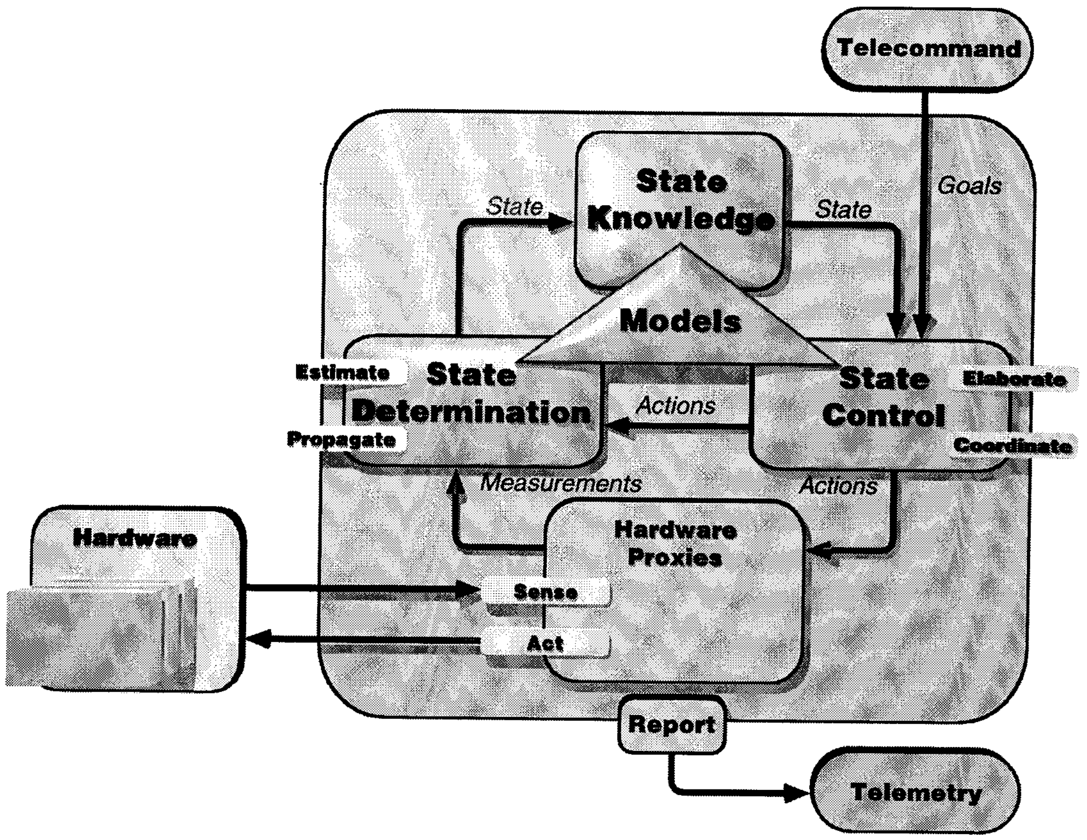


Figure 1. This diagram emphasizes several MDS architectural themes: the central role of state knowledge and models, goal-directed operation, separation of state determination from control, and closed-loop control.

the system's history, expectations, and plans. As well as providing the fundamental coordinating mechanism on the spacecraft, state timelines are also the objects of a uniform mechanism of information exchange between flight and ground.

Explicit Use of Models

MDS expresses domain knowledge explicitly in inspectable models rather than implicitly in program logic. The models separate the application-specific knowledge from the reusable logic for applying that knowledge to solve a problem. A model built for a specific mission may also be reused on similar future projects. The task of customizing MDS for a mission, then, becomes largely a task of defining and validating new or reused models.

Join Navigation, Attitude Control, and Robotics

MDS builds navigation, attitude control, and robotics from a common mathematical base. They have been built in the past as separate development efforts because they operate over different timescales or in different environments, or because their dynamics don't greatly affect each other. When they need to share information, in cases such as maneuver execution or pointing towards celestial bodies, the interfaces are ad hoc and conversions are needed between different forms of knowledge representation.

In general, elements that work by themselves often fail to work together when they are built separately. Separately engineered applications create an environment that makes it easy for bugs to slip through the cracks simply because of the combinatorics of the interactions involved. This is a major source of unreliability. Upcoming missions will have greater needs to share more information between disciplines that have been separately engineered in the past. For example, rovers will need to navigate to targets identified by orbiting mapping spacecraft and point antennas to Earth or orbiting relay spacecraft. Docking missions and missions to orbit small planetary bodies will require tight coupling between attitude control and navigation. The MDS solution is to build subsystems from common architectural elements, rather than the other way around. In this way, the interactions are more reliable by using common interaction management mechanisms and common forms for knowledge representation.

3. State

State Variables

The MDS architectural element for representing state is the *state variable* [2]. All users of state knowledge get it from state variables. Only one version of a state's estimate is represented and it is stored in a single state variable to discourage potentially inconsistent private estimations. A computer, however, may need a copy of a state estimated on a different computer. In this case, the estimating computer

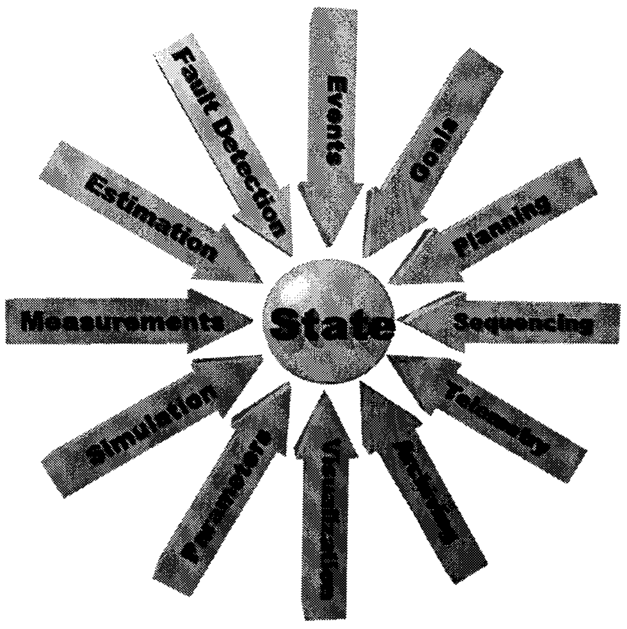


Figure 2. System state is the architectural centerpiece for information processing. *State* is a representation of the momentary condition of an evolving system.

sends new estimates to the remote computer. The copy is accessed just like the original except it prevented from being updated by a local estimator. A state variable that that is the result of this mirroring process is called a *proxy state variable*. A state variable that is updated locally with an estimator is called a *basis state variable*. A state's estimate is the system's best guess of the "true" physical state that the estimate represents. MDS recognizes that state estimates are not "truth" and requires that all state estimates include an assessment of uncertainty. MDS stores the state history for state variables as functions of time in the state variable's *value history*. Because a state's value history implies its derivatives, MDS implicitly represents a state's derivatives in its value histories. If an application needs an explicit representation of a state's derivative, then MDS requires that the derivative be encapsulated as part of the state's value and requires that the application insures that the value history and the explicit representation are consistent. Some states are computed simply as functions of other states (i.e. their estimations do not incorporate measurements or commands). Such states are called *derived state variables.* This concept comes in handy later during the description of graph state variables.

Absolute and Relative States

Many states are *absolute* in nature. Some examples of absolute states are the electrical current flowing through a heater, whether a switch is open or closed, the deployment of a parachute, whether a pyrotechnic device is fired or not, whether a cable is cut, propellant volume and pressure, the power state of an instrument, and spacecraft mass[2]. Many key spacecraft states, however, are not absolute quantities that correspond exactly to measured states. For example, the spacecraft position is relationship between its location and another (e.g. the Earth, the Sun, the Mars Barycenter, a target of opportunity, etc.). One cannot talk about location without defining an originating point of reference and a coordinate frame. Similarly, a spacecraft orientation state is meaningless without also specifying its coordinate frame. Another example is voltage, which expresses an electrical potential between two objects. In addition, the flow of information between directly connected subsystems in a network is defined in terms of the two nodes. Even heat flow in a thermal model is defined in terms of specific heat flows between pairs of nodes.

Many times there is a need to compose such relative states to derive new relationships. For example, if the voltage for a grid in an instrument is measured with respect to the instrument's ground voltage, and the instrument ground is offset from the spacecraft ground, then the grid's voltage with respect to the spacecraft is the sum of the two voltages.

A rover example demonstrates a more elaborate need for modeling composite relationships (see figure 3). A rover needs to autonomously control the relationship (**d**) of the position of a rover's end effector (**D**) relative to a rock on the Mars surface (**E**). This relationship state may be composed from the following constituent measured relationship states: the relative locations of the effector relative to the forearm (**D** *w.r.t.* **C**), the forearm relative to the upper arm (**C** *w.r.t.* **B**), the upper arm relative to the rover body (**B** *w.r.t.* **A**), and the rover body relative to the rock (**A** *w.r.t.* **E**). Changes to any of the constituent states should be reflected in the composite relationship state value for the end effector's position. Also, the end effector's proximity sensor directly provides an accurate measurement of the rock location when the effector is close to the rock. The position of the end effector should use the proximity sensor when it is a more accurate estimate than the lengthy derivation **D->C->B->A->E**. Thus, the selection of states used in the derivation of a composite state's value may need to depend on a property of the derivation, namely its accuracy. The rover's location relative to a landing site is composed of a sequence of location offsets between waypoints. Each offset between two waypoints is a constituent relationship of the rover's location state. The number of offset states increases as the rover identifies and crosses the surface from waypoint to waypoint. The derivation for the rover's location should incorporate these new offset states as the mission progresses. Thus, the derivation of a composite relationship state should incorporate new constituent states as they are created. Finally, the measurement device for each joint angle in the rover's arm may share an analog-to-digital converter that introduces the same bias error in each of the measurements.

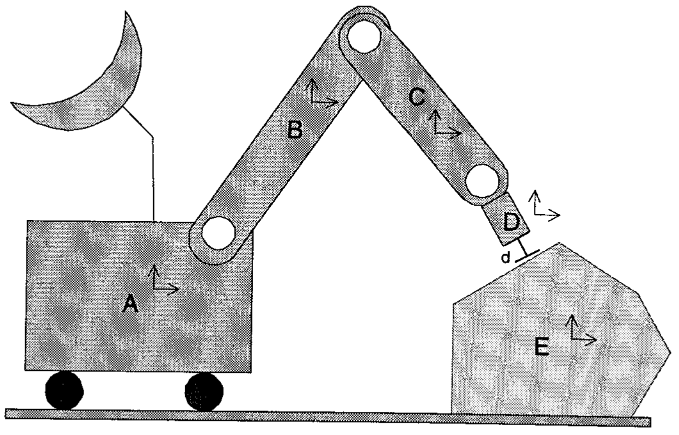


Figure 3. Rover arm example.

Estimation with these measurements produces a set of correlated estimates for the angle states. The uncertainty of the end effector's position may be dependent on the angle correlations. Thus, the derivation of a composite state should use the dependencies (i.e. correlations) between its constituent states.

These examples demonstrate a general need for identifying, organizing, and estimating composite states that represent relationships between objects.

4. Relative State Representation Today

Spacecraft represent relative states usually in an implicit rather than explicit manner. For example, voltages are represented as offsets, as it is understood that these offsets are with respect to the spacecraft chassis ground. Also, there is typically no systematic representation of uncertainty in current space applications.

Spacecraft attitude control systems typically estimate the relative state for spacecraft orientation using onboard sensors that measure spacecraft rotation rates, identify stars, and directions to the sun or nearby planetary bodies. Spacecraft orientation and directions to solar system bodies are implicitly defined with respect to a coordinate frame reference (e.g. an inertial coordinate frame such as EME 2000). In many missions, there is no explicit representation in the onboard attitude control system of the spacecraft trajectory, or the orbits and rotations of the planetary bodies. In these cases, the ground navigation system computes these quantities using detailed ground based models and converts them into simplified propagation models that approximate the directions in the coordinate frame that the spacecraft implicitly understands. These simplified propagation models are uplinked to the onboard attitude control system. It in turn uses them to produce directions at needed times for orienting the spacecraft, pointing its antennas and instruments, and avoiding harmful or disruptive radiation sources.

[2] Mass may be considered a relative state at relativistic velocities.

The pointing system model that resides in the Cassini spacecraft attitude control system improves on this situation [3]. It explicitly represents the orbits of planetary bodies, the spacecraft trajectory, and orientations between coordinate frames. Translations between planetary bodies and the spacecraft, and orientations between coordinate frames are organized as edges in a tree. Each edge either represents a relative position or orientation between nodes. The relative positions are vectors between two location nodes (e.g. a planetary body or the Cassini spacecraft.) The relative orientations are quaternions between two coordinate frame nodes (e.g. EME 2000 and the spacecraft local coordinate frame). Some nodes stand in for both locations and coordinate frames. In this way, the direction of a planetary body relative to the spacecraft is computed by first finding the path in the tree between the body and spacecraft. The direction is computed by adding the sequence of vectors and applying the rotations along the path. A found path can be reused for computing a direction at a later time without having to search the tree. The onboard pointing system propagates the relative locations in the tree edges using uplinked conic or polynomial functions. These functions as well as functions for varying orientations are commanded by the ground and are computed from more detailed ground based navigation models.

JPL's Deep Space 1 mission has an onboard navigation application called AutoNav that also contains a model of the spacecraft trajectory; however, it can autonomously update the model of the trajectory using asteroid sightings. AutoNav explicitly represents and propagates trajectory and orbit models within the navigation software, but in a form that is not directly understood by attitude control. Attitude control queries AutoNav for a direction to a planetary body, and AutoNav returns a direction that is implicitly in the EME200 coordinate frame. Attitude control rotates this vector into the spacecraft coordinate frame using its estimate of the spacecraft's attitude. AutoNav computes the direction by propagating a polynomial function for the direction. When the flyby geometry changes too fast for repeated AutoNav queries, AutoNav provides attitude control with a first order approximation of the dynamics of the changing direction (an initial vector and a velocity vector). In this case, attitude control rather than AutoNav propagates the direction. AutoNav receives models of planetary orbits and the spacecraft trajectory uplinked from the ground in the form of ephemeris files containing the polynomial coefficients. The ground software creates these by reducing a yet more detailed orbit representation in the ground navigation system. These more detailed ground representations are composed of sets of trajectory, ephemeris, and orientation files that represent relative location and orientation states. These relative locations and orientations are defined in terms of a set of coordinate frames and fixed locations corresponding to the spacecraft and planetary bodies. The relations are organized using a tree topology, where each coordinate frame or location is identified as a node in a tree. Relative translations and rotations are organized as edges between the nodes in a tree. One tree contains all the translations and another contains all the rotations. These translations and rotations can be composed to derive new relative states by following paths within the trees.

The Mars Exploration Rover is planning on using a set of frames for translating between the previous and current rover location and orientation and the next target location. The rover's arm also has a system for representing the relative states between its elements and science targets.

Common to these applications is the observation that only a single derivation is allowed because of the use of tree topology to represent relationships. In addition, there is lack of systematic representations for the relationships between the various applications, and in different systems within the same mission. Further there is no explicit representation of uncertainty in the applications' software architecture. MDS graph state variables provide the same functionality as these applications while also overcoming their shortcomings.

5. MDS GRAPH STATE VARIABLES

MDS graph state variables (GSVs) provide a general graph-based state representation for expressing relative states and their uncertainties in an explicit fashion. The states can be rotations between coordinate frames, relative locations of planetary objects and spacecraft, directions to targets, locations of rover arm elements and other articulated elements. Temperatures of devices not directly measured can be modeled by graph state variables, where the relative states are thermal flow between directly measured or estimated heat sources. Another application are the gains in a telecommunications network. More simply, graph state variables can model the connectivity and communications rates of onboard networks of communications buses or between orbiting communication satellites and ground stations. Graph state variables may also be used to represent relative voltages.

Nodes and Relationships

A graph state variable knows about a set of uniquely identifiable *nodes*. Each node corresponds to an object that has a *direct relationship* with another node in the graph state variable. The direct relationship is a relative state and is represented in the software as a state variable. In the previous rover arm example, the nodes are the end effector (**D**), the forearm (**C**), the upper arm (**B**), the rover body (**A**), and the location of the rock (**E**). Each node is a frame. A *frame* consists of a reference coordinate frame with its origin corresponding to the location of an object. These nodes can be seen in figure 4. Each direct relationship is a directional transitive relative state. In this example, the relationship is a *6 degree of freedom* (6 DOF) transformation. A 6 DOF transformation contains both a translation and a rotation. Thus, each relationship represents a combined relative location and orientation state. For example, the relationship *d* is the translation and rotation from the end effector (**D**) to the rock (**E**). Each node in the rover example, therefore,

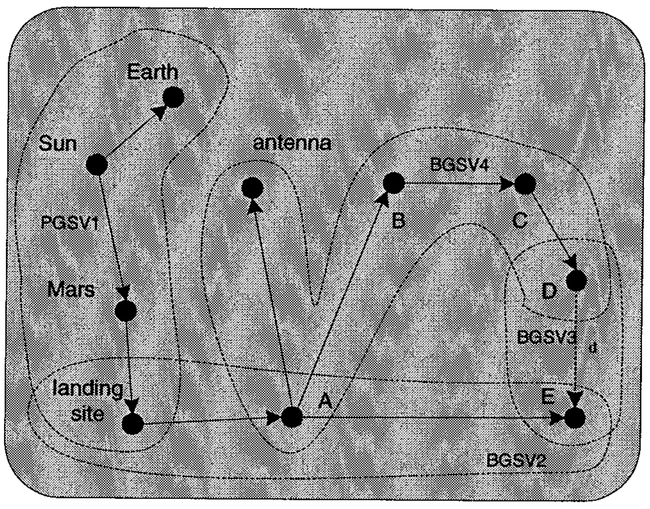


Figure 4. Rover arm example graph state variable.

corresponds to a location *and* a coordinate frame. If two of the nodes share the same coordinate frame, for example the Mars surface location and the rock, then the 6 DOF direct relationship between is a translation with a null rotation.

Relationships and Derivations

A graph state variable can compute the relative state between two of its nodes. The nodes need not have a direct relationship between them. In other words, they may be non-adjacent. A GSV can derive a relative state's value for non-adjacent nodes in the GSV by concatenating the relationships along a path between the two nodes. A *path* is defined as a sequence of non-repeated direct relationships. GSVs require that a relative state relationship type be symmetric and transitive. For a relationship type to be symmetric it must be invertible. If a relationship value from (**A**) to (**B**) exists, then so does a relationship value from (**B**) to (**A**). For a relationship type to be transitive, an instance of it concatenated with another instance must also be an instance of the relationship type. For example, if a direct relationship instance R_{AB} exists from (**A**) to (**B**), and if another R_{BC} exists from (**B**) to (**C**), and both are instances of a transitive relationship, then a *derived relationship* instance R_{AC} exists for the relationship from (**A**) to (**C**). The derived relationship is computed using a concatenation operator (•). Thus R_{AC} equals $R_{AB} \bullet R_{BC}$.

In the rover example above, the relationship type, a 6 DOF transformation, is symmetric and transitive. The relative state for the end effector's (**D**) position and orientation with respect to the rock (**E**) is a derived relationship. It's derivation is the concatenation, and inversion where needed, of the direct relationships R_{CD}, R_{BC}, R_{AB}, and R_{AE}. More explicitly, a derivation for R_{ED} is $(R_{CD})^{-1} \bullet (R_{BC})^{-1} \bullet (R_{AB})^{-1} \bullet R_{AE}$.

Note that a direct relationship may itself be a derived state variable but its derivation is not restricted to be computed as a GSV relationship derivation. For example, the rover's traversability between locations on a map may be represented as a sequence of moves within a grid. Each move may have a level of traversability. The traversability for any particular segment may be computed from an elevation map state that is estimated from camera observations. It would make no sense to represent the traversability of each possible move. The number of direct traversability relationships would be very large. Further, most are eventually discounted as being to difficult to cross or in the wrong direction. One approach would be to have the traversability not directly represented as states, but only computed as need from a state that represents the whole grid of elevations. In this case, a GSV could represent traversability between points by referencing a derived state that computes traversability from the elevation map state.

Alternative Derivations

A graph state variable can select between alternative derivation paths. A graph can have loops that produce multiple paths between two nodes. This is useful when there may be more than one way to derive a relative state. In this case, an application provides a *path arbitration criterion* that the graph state variable uses to select between more than one possible derivation. If the user needs only reasonable derivations, the adapter can provide a *pass / fail criteria*. In this case, the graph state variable will only return derivations that pass the criteria. If a user needs only the *n* "best" derivations, then the adapter can provide a *utility function* that the GSV uses to return the *n* derivations that have highest utility.

The rover arm example above has two possible derivation paths for the distance, **d**, between the rock (**E**) and the end effector (**D**): **E->D** and **E->A->B->C->D**. The GSV should return the distance directly sensed by a proximity sensor on the end effector (**E->D**) when it is close to the rock. In this case the discriminating factor could be the derivation's uncertainty. The directly sensed distance may have a much smaller uncertainty that the indirect long derivation. The utility function would simply return the certainty for each derivation passed to it. The GSV would choose the derivation that has the greatest utility (e.g. certainty).

Graph state variables also allow for the use of both a utility function and a pass / fail criteria together. In this way an adapter can have the GSV return the *n* best derivations that muster a pass / fail criteria.

Basis and Proxy Graph State Variables

A GSV is composed of a collection of nodes and the direct relationships that connect them. Each direct relationship is stored in a state variable. GSVs that are composed of basis and/or derived state variables are called *basis graph state variables* (BGSVs). Basis graph state variables contain relationships that are estimated in the local deployment. The proxy pattern that was first established for state variables is repeated here for GSVs. A *proxy graph state variable* (PGSV) is composed of direct relationships that are estimated on a different processor in a separate deployment. In this case, each direct relationship is stored in a proxy state variable.

In the rover example, the rover estimates its location (**A**) in reference to the landing site, and the location of the rock (**E**) relative to the location of the rover in basis graph state variable BGSV2. Similarly, it estimates the 6DOF transformations for its articulated antenna and upper arm relative (**B**) to the location and coordinate frame of the rover body (**A**) and stores the estimates in BGSV4. The 6DOF transformations between the rover arm elements are also estimated by the rover and stored in BGSV4. All the transformations for the articulated elements are stored in the same BGSV because they are correlated. Correlated relationships will be discussed further in the dependencies section below. Finally, the rover's 6DOF estimate of the rock (**E**) relative to the end effector (**D**) using the proximity sensor is stored in BGSV3. What remains are the relationships that are estimated on the ground and copied into proxy graph state variable PGSV1. These relationships include orbit trajectories for the Earth and Mars, the location of the Sun, and the location of the landing site on Mars relative to the Mars frame.

Aggregation and Derived Graph State Variables

A GSV may be composed of other GSVs. Such a GSV is called a *derived graph state variable* (DGSV). A DGSV can derive new relationships from the relationships that are in its various constituent GSVs. Common nodes that appear in more than one of the constituent GSVs provide the bridge between them. This allows relationship derivation paths to extend over different GSVs. The derived GSV can link sets of direct relationships that are estimated in both local (Basis GSVs) and remote (Proxy GSVs) deployments. Further, the derived GSV allows for a hierarchical organization of sets of relationships. A derived GSV is not limited to be only composed of BGSVs or PGSVs. It may have as constituents other derived GSVs. A hierarchy of collections of relationships can be created, by having DGSVs, within DGSVs, within DGSVs, etc.

The rover example has a single derived graph state variable called the frame derived graph state variable. It is composed of 4 constituent GSVs: PGSV1, BGSV2, BGSV3, BGSV4. The landing site node links the proxy graph state variable for ground estimated navigation 6DOF states (PGSV1) and the rover estimated navigation 6DOF states (BGSV2). The rover body node (**A**) links the rover estimated navigation states with the 6DOF relationships between the articulated elements (BGSV4). Finally, BGSV3 is a bit of a special case. It contains the relationship for the directly measured distance between the end effector (**D**) and the rock (**E**). Thus, it provides the link that allows the estimate of the rock's location (**E**) using navigation and vision algorithms (BGSV2) to be compared with the estimate using kinematics (BGSV4) and the proximity sensor (BGSV3).

Dependencies

GSVs can represent the *dependencies* between its relationships. GSVs use dependencies to compute the uncertainties of derived relationships. Dependencies represent correlations between the direct relationships in a graph state variable. A GSV may reference a state variable

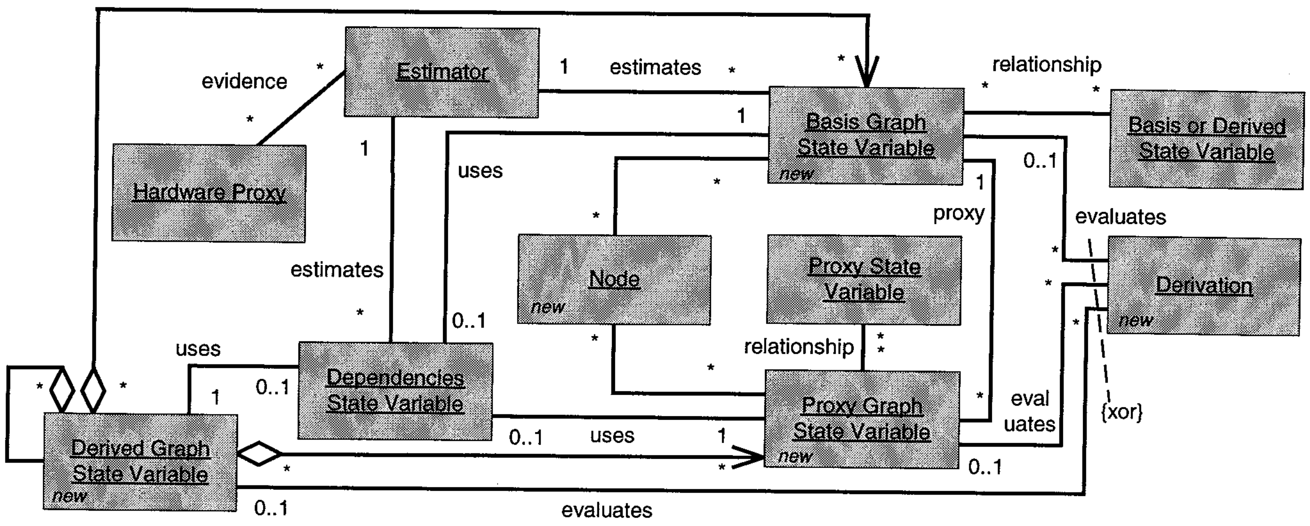


Figure 5. Unified Modeling Language (UML) of the graph state variable architectural elements.

that contains the dependencies. Suppose two direct relationships in a GSV are A and B, and each has an uncertainty that is represented as a normal distribution with variances σ_A^2 and σ_B^2, respectively. The dependency in the GSV is the covariance of A and B, $cov(A,B)$. If the concatenation of the two states is the subtraction operator, then A•B is equal to A-B and the GSV computes the uncertainty of A•B as $\sigma_A^2+\sigma_B^2-2cov(A,B)$.

The rover arm example contains 3 direct relationships in BGSV4 that represent relative locations and orientations of the rover body, upper arm, forearm, and end effector. Each direct relationship is estimated using measurements from different potentiometers, each of which measures a particular joint angle. The same A-to-D converter, however, digitizes the voltages from all potentiometers. A bias in the A-to-D converter affects all conversions. Thus, each estimate of a rover arm direct relationship is offset by the bias and is correlated with the other rover arm direct relationships.

Dynamic Topology

Some applications need to add new relationships and nodes as new objects of interest become pertinent to the mission. Examples include science targets identified by a rover along the surface of Mars, unplanned spacecraft flybys of opportunistic targets of interest, and asteroids added for optical navigation purposes. In some cases, the topology of the set of relative states changes without adding new nodes. For example, spacecraft position may be estimated with respect to the Earth immediately after launch. Later during interplanetary cruise, it is estimated with respect to the Sun (or the Solar System Barycenter). Finally, it may be estimated with respect to a planetary body for an upcoming flyby or orbit insertion. All of these objects are typically identified and tracked prior to launch; however, new direct relationships of the spacecraft with respect to nearby objects need to be established. These examples show there is need for graph state variables to accommodate topology changes that add new nodes and relationships.

To meet these necessities, graph state variables include methods for adding nodes and new direct relationships. The underlying MDS software architecture provides for adding and connecting new state variables that correspond to the new direct relationships. Path arbitration criteria can be applied to use new direct relationships rather than previously established paths.

Performance

The search for a path between two nodes may be too slow for applications containing large graphs or high rate queries. Graph state variables have a number of features that allow an adapter to reduce the time for deriving a relationship.

Cached Paths—Graph state variables provide a capability for returning derived relationships in a small amount of time for frequent repeated queries. If the derivation is frequently along the same path, an application may save the path returned by a query. The path may be used in a subsequent relationship query to avoid the overhead of searching the graph. Saved paths are automatically invalidated and re-evaluated when a direct relationship along the path or the graph's topology changes.

Domain Specific Search Algorithms—In addition to providing generic search algorithms for finding paths, graph state variables allow an application to provide its own search algorithm. An application can use knowledge of its domain in its GSV search algorithm to find a path between nodes.

Search Algorithm Hints—Graph state variables provide a capability that allows an application to provide search hints. A hint can be specified in a relationship query and used by the search algorithm to direct searches. A hint may take a variety of forms. For example, the search algorithm may use a path arbitration criterion as a search hint. Another option for an application is to provide a partial path screening criterion. The search algorithm uses it to eliminate candidate partial paths during each step of a graph traversing search algorithm. A partial path screening criterion is like a pass / fail path arbitration criteria and it applies to partial paths. In addition, the criterion may be a function of the latest relationship and/or node on the partial path. Another type of hint is a partial path prioritization function. The search algorithm uses it to select the next path explored during each step of a graph traversing search algorithm. A partial path prioritization function is like a path arbitration criteria utility function and it applies to partial paths. It may also be a function of the latest relationship and/or node on the partial path.

6. MARS EDL EXAMPLE

A 6DOF adaptation of graph state variables for a Mars entry, descent, and landing prototype is depicted in figure 6. The spacecraft attitude and location is a relative state that represents the translation and rotation of the spacecraft basebody frame with respect to the Mars-centered inertial frame. The Mars-centered Mars-fixed frame is a rotating frame with respect to the Mars-centered inertial frame. The Mars surface fixed frame is the initially targeted landing site and is related to the Mars-centered Mars-fixed frame by a location offset from the center of Mars, and by a rotation that has the Z-axis vertical and X and Y axes along Mars longitude and latitude lines. The re-target frame is a location offset from the Mars surface fixed frame used for hazard avoidance during landing.

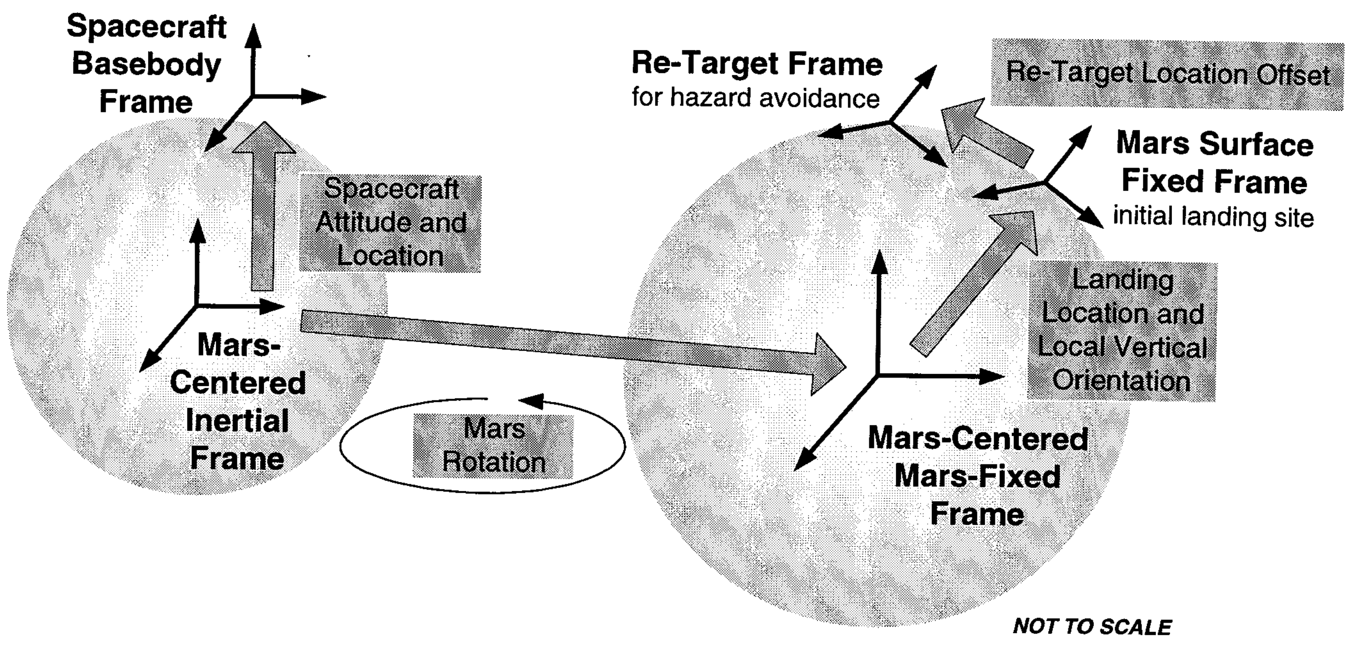

Spacecraft Basebody Frame
Spacecraft Attitude and Location
Mars-Centered Inertial Frame
Mars Rotation
Re-Target Frame
for hazard avoidance
Re-Target Location Offset
Mars Surface Fixed Frame
initial landing site
Landing Location and Local Vertical Orientation
Mars-Centered Mars-Fixed Frame
NOT TO SCALE

Figure 6. 6 degree of freedom frames and relationships for a Mars lander prototype.

In addition to these navigation frames, there are frames to represent the spacecraft center of mass and the location of spacecraft roll thrusters, descent engines, and an ideal sensor that measures the spacecraft location and orientation. A true flight application would contain separate frames for rotation measurement devices (e.g. gyroscopes), accelerometers, a star tracker, radar and lidar. These have been abstracted for the purposes of the prototype and will be added in future versions. The spacecraft center of mass frame relationship with the spacecraft basebody is a position offset represented as a 6 DOF relationship having no rotation. Similarly, the thruster and descent engine frames are defined in terms of relative states that are only translations with respect to the spacecraft basebody frame. Finally, the ideal sensor is just a stand in for more realistic sensors, so its frame's relationship with the spacecraft basebody frame is a constant identity transformation (no rotation or translation).

These frames and relationships are collected into 5 basis state variables within the frame derived graph state variable shown in figure 7. Each of the relationships is stored in a basis state variable. The thruster, descent engine, and ideal sensor frames and their relationships are gathered into the same basis GSV because these represent calibration parameters for device alignments. Such parameters are typically estimated on the ground and uplinked to the spacecraft and are collected into the same GSV. If they were actually uplinked from the spacecraft in the prototype, these relationships would be stored in proxy state variables and collected into the same proxy GSV. The spacecraft attitude and position is in its own basis GSV because it will be estimated on the spacecraft. The relationships between the Mars inertial and fixed frames are in the same basis graph state variable because they are usually estimated on the ground. Again, in a real application, these would also be proxies. The re-target frame position offset is within its own basis GSV because an onboard algorithm for hazard avoidance estimates it.

The thruster and descent engine force direction states are defined as unit vectors in their individual frames. The torque for roll thruster "i" is computed by the following formula:

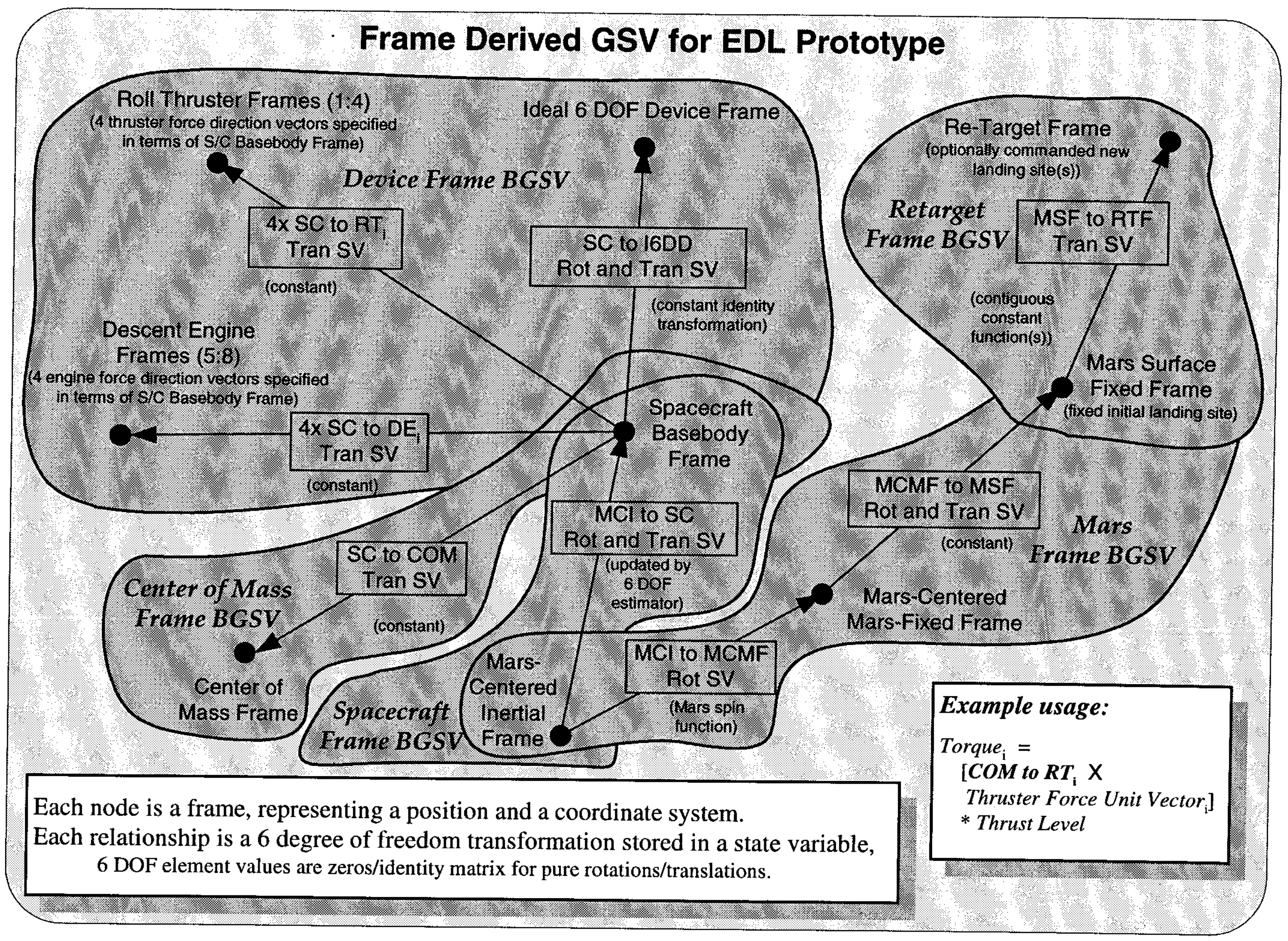
$\text{Torque}_i =$
$[\text{COM to RT}_i \text{ X Thruster}_i \text{ Force Direction}] * \text{Thrust Level}_i$

The descent engine torques are computed in the same fashion. The spacecraft moments of inertial is another state that is defined in terms of a frame. In this case it is the center of mass frame. This along with the stae for the spacecraft mass defines the spacecraft inertial properties.

7. Systems Engineering Relative States

MDS state analysis is a model-based process to aid systems and software engineering. State analysis prompts a methodical and rigorous analysis of systems engineering concepts in terms of MDS architectural elements. These architectural elements have a one-to-one correspondence to specific MDS software frameworks. In this way, there are fewer chances for the software implementation to misinterpret systems engineering requirements.

Graph state variables provide both a systems engineering methodology for explicitly describing needed relative states and an unambiguous a way for transforming the needs into elements in a software architecture. The systems engineer describes a relationship and the objects between which it


Frame Derived GSV for EDL Prototype
Roll Thruster Frames (1:4)
(4 thruster force direction vectors specified in terms of S/C Basebody Frame)
Ideal 6 DOF Device Frame
Device Frame BGSV
4x SC to RT$_i$ Tran SV
(constant)
SC to I6DD Rot and Tran SV
(constant identity transformation)
Descent Engine Frames (5:8)
(4 engine force direction vectors specified in terms of S/C Basebody Frame)
4x SC to DE$_i$ Tran SV
(constant)
Spacecraft Basebody Frame
Re-Target Frame
(optionally commanded new landing site(s))
Retarget Frame BGSV
MSF to RTF Tran SV
(contiguous constant function(s))
Mars Surface Fixed Frame
(fixed initial landing site)
MCMF to MSF Rot and Tran SV
(constant)
Mars Frame BGSV
Mars-Centered Mars-Fixed Frame
MCI to SC Rot and Tran SV
(updated by 6 DOF estimator)
SC to COM Tran SV
(constant)
Center of Mass Frame BGSV
Center of Mass Frame
Spacecraft Frame BGSV
Mars-Centered Inertial Frame
MCI to MCMF Rot SV
(Mars spin function)

Each node is a frame, representing a position and a coordinate system.
Each relationship is a 6 degree of freedom transformation stored in a state variable,
6 DOF element values are zeros/identity matrix for pure rotations/translations.

Example usage:

Torque$_i$ =
[***COM to RT***$_i$ X
Thruster Force Unit Vector$_i$]
* *Thrust Level*

Figure 7. Graph state variables for a Mars lander prototype.

relates. This corresponds to an adapted software type for a GSV relationship and the nodes in a graph state variable instance. The system engineer identifies the direct relationships that will be estimated, proxied from another deployment, or derived from other states. Each of these is instantiated in the software as a basis, proxy, or derived state variable, respectively. The system engineer groups the direct relationships that are estimated together into basis graph state variables. Each grouping is created in the software as a basis graph state variable. The system engineer groups the direct relationships copied from other deployments into proxy graph state variables. Similarly, each of these groupings is built into the software as a proxy graph state variable. The system engineer collects the graph state variables having the same kind of relationship into a derived state variable, which is manifested as such in the software. Finally, the system engineer defines those objects that are defined in terms of a GSV node, and need to be expressed in terms of another. The software creates these objects and methods to transform them using an instance of

8. Conclusions

The MDS architecture and graph state variables are a foundation for spacecraft software that can meet the ambitious autonomy challenges of future space exploration missions. Graph state variables meet key MDS architectural themes to explicitly model states and join attitude control, navigation, and robotics. Graph state variables provide a common reusable framework that will allow easy management of attitude control, navigation, and robotics interactions and provide for shared systematically engineered data. GSV representations provide a common mathematical base for spacecraft orientation, location, trajectories, dynamics, and kinematics. In addition, they have potential uses in the power, telecommunications, and networking domains. Graph state variables explicitly model relative relationships between objects as relative states within graphs. They provide an unambiguous and systematic translation of system engineering requirements on relative states into a reliable software implementation

In summary, graph state variables:

(1) explicitly represent relative states as relationships between node objects,
(2) explicitly model the composition of derived relative states from elemental relative states,
(3) combine multiple sets of relative states (sub-graphs) estimated separately or copied from remote locations,
(4) produce different results for different derivation paths and can select between them,
(5) can change the graph topology to add node objects and relative states,
(6) represent dependencies between relative states (e.g. correlations).

9. ACKNOWLEDGEMENTS

The research and design described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

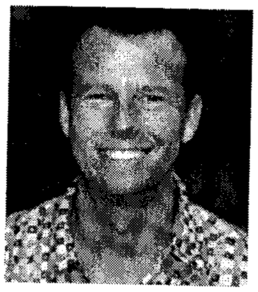
The authors thank the entire MDS team at JPL for their contributions, without which this work could not have been done. He is exceptionally appreciative of the following MDS team members for our valuable graph state variable discussions: Daniel Dvorak, Sandy Krasner, Michael Lisano, Steve Peters, Nicolas Rouquette, and Marcel Schoppers.

REFERENCES

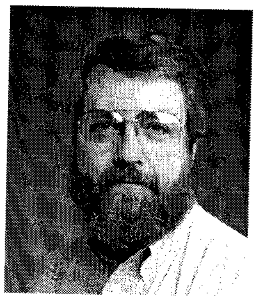
[1] D. Dvorak, R. Rasmussen, G. Reeves, A. Sacks; "Software Architecture Themes in JPL's Mission Data System," *2000 IEEE Aerospace Conference Proceedings,* March 2000.

[2] D. Dvorak, R. Rasmussen; "State Representation in the Mission Data System," *2002 IEEE Aerospace Conference Proceedings,* March 2002.

[3] R. Rasmussen, G Singh, D. Rathburn, G. Macala; "Behavioral Model Pointing on Cassini Using Target Vectors," *18th Annual AAS Rocky Mountain Guidance and Control Conference,* February 1995.



Matthew Bennett *is a senior engineer in the Avionics Systems Engineering section of the Jet Propulsion Laboratory, California Institute of Technology. He has researched spacecraft autonomy and has broad experience in spacecraft development, test, and operations, including the Galileo Jupiter and Cassini Saturn orbiter missions. He has developed mission software for fault protection, guidance and control, performance analysis, and simulation. He holds an MS from the University of Washington in Computer Science, and a BS from the University of California at San Diego in Computer Engineering.*



Robert Rasmussen *is a principal engineer in the Information Technologies and Software Systems division of the Jet Propulsion Laboratory, California Institute of Technology, where he is the Division Technologist and the Mission Data System architect. He holds a BS, MS, and Ph.D. in Electrical Engineering from Iowa State University. He has extensive experience in spacecraft attitude control and computer systems, test and flight operations, and automation and autonomy — particularly in the area of spacecraft fault tolerance. Most recently, he was cognizant engineer for the Attitude and Articulation Control Subsystem on the Cassini mission to Saturn.*